

Recursive Schur Decomposition*

Rahul S. Sampath[†], Bobby Philip, Srikanth Allu and Srdjan Simunovic
 Computer Science and Mathematics Division
 Oak Ridge National Laboratory
 One Bethel Valley Road, Oak Ridge, TN 37831

October 24, 2012

Abstract

In this article, we present a parallel recursive algorithm based on multi-level domain decomposition that can be used as a preconditioner to a Krylov subspace method to solve sparse linear systems of equations arising from the discretization of partial differential equations (PDEs). We tested the effectiveness of the algorithm on several PDEs using different number of sub-domains (ranging from 8 to 32768) and various problem sizes (ranging from about 2000 to over a billion degrees of freedom). We report the results from these tests; the results show that the algorithm scales very well with the number of sub-domains.

1 Introduction

Domain decomposition methods are a class of numerical techniques that are used to solve large scale boundary value problems on complex domains using several processors. These algorithms are based on the classic principle of “divide and conquer”. In these methods, the domain of interest is divided into a number of sub-domains and the global problem is reduced to a smaller problem defined only on the interfaces between the sub-domains. The solution to the interface problem provides the necessary and sufficient

*Notice: This manuscript has been authored by UT-Battelle, LLC, under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

[†]Corresponding author: Rahul S. Sampath (sampathrs@ornl.gov).

conditions required to split the global problem into a set of independent problems defined on each sub-domain. These sub-problems are easier (with respect to mesh generation) and cheaper (with respect to storage and operation count) to solve compared to the global problem; moreover, they can be solved concurrently. For a good review on different domain decomposition methods, we refer the reader to [22, 21, 14] and references therein.

The technique of domain decomposition can be applied to both linear and nonlinear problems but, we only focus on linear problems in this article. In particular, we restrict our attention to sparse linear systems of equations that are generated by discretization and perhaps linearization of partial differential equations (PDEs). Furthermore, we are only concerned with discretizations using the finite element method (FEM) although, the methods discussed here could potentially be applied to other types of discretizations as well.

The interface problem forms the core of any domain decomposition method. It is typically solved (possibly only approximately) using an iterative algorithm because using a direct method would be too expensive. In fact, the matrix (required for a direct method) corresponding to the interface problem is seldom constructed explicitly as it would involve solving the sub-domain problems several times. A major challenge faced by domain decomposition methods is the fact that the size of the interface problem grows as the number of sub-domains increases. For single-level domain decomposition methods, an increase in the interface problem size translates to an increase both in the number of iterations required to solve the problem as well as the cost per iteration; this critically limits the scalability of these methods with respect to the number of sub-domains. Two-level domain decomposition methods [6, 7, 12] attempt to address this issue by solving an additional coarse global problem, known as Coarse Grid Correction, coupling all the sub-domains at each iteration. This makes the number of iterations (equivalently, the convergence rate of the algorithm) required to solve the problem independent of the number of sub-domains but, the cost of applying the Coarse Grid Correction grows with the number of sub-domains. Besides, the Coarse Grid Correction is problem dependent and it is not straightforward to apply these methods to solve arbitrary PDEs; these methods have mostly only been applied to solve structural mechanics problems. Multi-level domain decomposition techniques [10, 8] take a different approach - they construct a hierarchy of interface problems using hierarchical domain decomposition such that the interface problem size at any level of the hierarchy is independent of the total number of sub-domains. These interface problems are nested, i.e., to solve the interface problem at any level

of the hierarchy we need to solve the interface problems at the level immediately below it. It would be too expensive to solve these interface problems exactly; hence, an incomplete factorization is used instead. Due to these approximations, the performance of these methods depends on the number of levels in the hierarchy. As the number of subdomains increases so does the number of levels in the hierarchy; this typically results in a deterioration in the performance of these algorithms.

In this work, we use a different technique to address the problem described above. Our algorithm uses a hierarchical domain decomposition framework as well but, it is distinct from existing multi-level methods in many respects. Presently, we have only applied this method to solve two dimensional PDEs on long and thin domains. This type of computational domain configuration is common in many problems of significant practical importance. For example, it is used to model oil pipelines [11, 18], electric power lines [23], nuclear fuel pins [17, 16] and arteries [9, 13].

The next section describes how we construct our hierarchical domain decomposition. Our multi-level solver is presented in Section 3. In Section 4, we report on some numerical experiments performed using this algorithm. The paper ends with some concluding remarks.

2 Hierarchical Domain Decomposition

Consider a domain whose length is much greater than its width. As mentioned earlier, we only focus on such domains in this paper. Let us suppose that the domain is divided along its length into a number of non-overlapping sub-domains. Also, assume that the number of sub-domains is a power of two; this assumption is made mainly for ease of exposition and implementation.

We organize the sub-domains into a nested hierarchy using a binary tree data structure [5]. Each node of the tree has at most two child nodes. The root of the binary tree is at level 0 of the tree and the children of any node in the tree are at one level greater than that node's level. The height of the tree is the greatest level in the tree.

The root of the tree represents the entire domain. First, we divide the domain equally along its length into two non-overlapping sub-domains. These sub-domains form the left and right children of the root. Each of these sub-domains is further split into two non-overlapping sub-domains in a similar manner; these sub-domains form the next level of the tree. The procedure is applied recursively until a pre-determined level. The sub-domains corre-

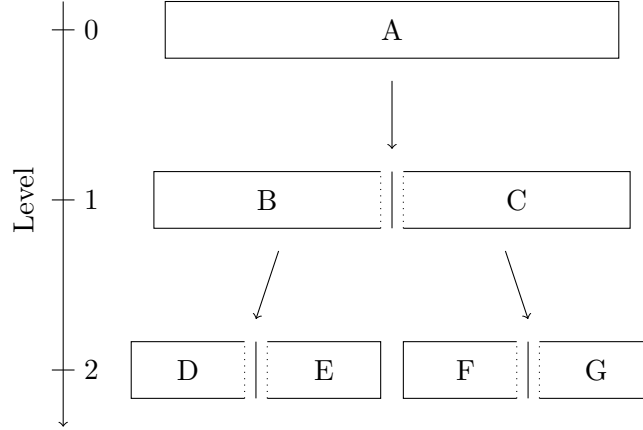


Figure 1: Binary tree based domain decomposition. A is the root of the tree that represents the full domain. B and C are pseudo sub-domains. D, E, F and G are the true sub-domains. The height of the tree is 2.

sponding to the leaves (nodes at the last level) of the tree are the “true” sub-domains that we started out with. The sub-domains corresponding to the other nodes of the tree are “pseudo” sub-domains. Each interior (non-leaf) node of the tree also represents an interface between two sub-domains. Figure 1 illustrates this hierarchical domain decomposition.

3 Recursive Schur Decomposition Preconditioner

Now, we will describe how we use the hierarchical domain decomposition presented in Section 2 to solve a sparse linear system of equations represented as: $Ku = f$; K is a sparse matrix, f is the right hand side vector and u is the unknown solution vector. In this paper, we will focus on systems of equations that are generated by applying the finite element method to discretize linear (or linearizations of nonlinear) PDEs. We solve this problem using a Krylov subspace method (GMRES to be precise) [20] with a multi-level algorithm as preconditioner. We describe the algorithm first for the case when there are only two sub-domains (Section 3.1) and then for the general case involving several sub-domains (Section 3.2).

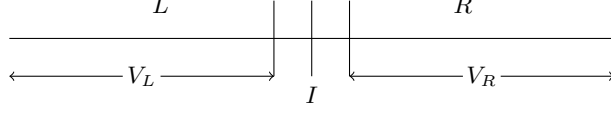


Figure 2: Decomposition with two sub-domains.

3.1 Two Sub-domains

Figure 2 shows a decomposition of a domain into two non-overlapping sub-domains. The sub-domain on the left is denoted by L and the one on the right is denoted by R . The interface between the sub-domains is denoted by I . The part of the sub-domain that remains after excluding the interface is denoted by V . The subscript on V refers to the corresponding sub-domain.

Using the above notation, we can write the system of linear equations as shown in Equation (3.1). The subscripts L , R and I denote the degrees-of-freedom corresponding to V_L , V_R and I , respectively. Note that $K_{II} = K_{II}^{(L)} + K_{II}^{(R)}$, where $K_{II}^{(L)}$ and $K_{II}^{(R)}$ are the contributions to K_{II} from L and R , respectively.

$$\begin{bmatrix} K_{LL} & 0 & K_{LI} \\ 0 & K_{RR} & K_{RI} \\ K_{IL} & K_{IR} & K_{II} \end{bmatrix} \begin{Bmatrix} u_L \\ u_R \\ u_I \end{Bmatrix} = \begin{Bmatrix} f_L \\ f_R \\ f_I \end{Bmatrix} \quad (3.1)$$

Our preconditioner is based on the corresponding Schur complement system [20], which is shown in Equation (3.2).

$$\begin{aligned} Su_I &= g \\ S &= K_{II} - K_{IL}K_{LL}^{-1}K_{LI} - K_{IR}K_{RR}^{-1}K_{RI} \\ g &= f_I - K_{IL}K_{LL}^{-1}f_L - K_{IR}K_{RR}^{-1}f_R \end{aligned} \quad (3.2)$$

Once u_I is known, u_L and u_R can be computed as shown in Equation (3.3).

$$\begin{aligned} u_L &= K_{LL}^{-1}f_L - K_{LL}^{-1}K_{LI}u_I \\ u_R &= K_{RR}^{-1}f_R - K_{RR}^{-1}K_{RI}u_I \end{aligned} \quad (3.3)$$

In our implementation, we invert the sub-domain matrices (K_{LL} and K_{RR}) exactly using a direct method. However, this is not necessary and one could use an approximate solver instead.

Note that we neither form S explicitly nor solve Equation (3.2) exactly; instead, we use a few iterations of a Krylov subspace method (GMRES to be precise) and compute an approximate solution (\widehat{u}_I). Then, we compute approximations to u_L and u_R by substituting \widehat{u}_I for u_I in Equation (3.3).

3.2 Multiple Sub-domains

Next, we turn our attention to a decomposition involving several sub-domains. We represent this decomposition using a binary tree as described in Section 2. We extend our algorithm from Section 3.1 to this case by applying it to level 1 of the tree. However, we don't use a direct method to invert the matrices (K_{LL} and K_{RR}) corresponding to the two pseudo sub-domains. Note that each of these pseudo sub-domains is itself a root of a sub-tree. So, we could compute an approximate solution to the linear systems involving K_{LL} and K_{RR} by applying the algorithm from Section 3.1 to level 1 of these sub-trees or equivalently level 2 of the original tree. This process can be applied recursively until we reach the last level (leaves) of the original tree. We can then invert the matrices (K_{LL} and K_{RR}) corresponding to the true sub-domains exactly using a direct method or, as mentioned earlier in Section 3.1, using an approximate solver.

Note that we need to invert (approximately) each of the matrices K_{LL} and K_{RR} at three steps in the algorithm: (1) computing $K_{LL}^{-1}f_L$ and $K_{RR}^{-1}f_R$ in Equation (3.2) and Equation (3.3), (2) computing $K_{LL}^{-1}K_{LI}v$ and $K_{RR}^{-1}K_{RI}v$ for some vector v for evaluating the matrix-vector product (MatVec) Sv , which is required at each iteration of the Krylov solver used to solve Equation (3.2) and (3) computing $K_{LL}^{-1}K_{LI}u_I$ and $K_{RR}^{-1}K_{RI}u_I$ in Equation (3.3). It will be too expensive to apply the overall preconditioner if we use the recursive process described above in each of these three steps.

We would like to point out that in the second and third steps K_{LL}^{-1} and K_{RR}^{-1} are always applied to a very sparse vector because most of the rows of K_{LI} and K_{RI} are zero. To be precise, the rows of this vector that correspond to nodes of the mesh that are not adjacent to a node on the interface are zero. We replace the recursive process in the second and third steps with a different approximation based on this observation. We continue to use the recursive process in the first step.

Let w be a sparse vector as described above, V represent some pseudo sub-domain and \widehat{V} represent the true sub-domain contained in V that shares the same interface as V . Consider the example shown in Figure 1: if V is B then \widehat{V} is E and if V is C then \widehat{V} is F. In the second and third steps, we approximate $K_{VV}^{-1}w$ as follows: (1) restrict w to \widehat{V} , (2) apply $K_{\widehat{V}\widehat{V}}^{-1}$ to the

restricted vector and (3) prolongate the result back to V by inserting zeros in the portion of the vector that does not belong to \widehat{V} .

3.3 Parallel Implementation

In our implementation¹, we assume that the number of true sub-domains is equal to the number of processors and we assign one true sub-domain to each processor. This assumption is made mainly for ease of exposition and implementation. Each interface is assigned to the processor that owns the sub-domain immediately to the left² of the interface.

The outermost GMRES solver used for the full system, the inner GMRES solver used for the Schur complement system at each level, and the direct solver used for the local problems on each true sub-domain are all standard components. We used the implementation provided in the PETSc package [2, 1, 3] for these components.

Algorithm 3.1: $S\text{-MATVEC}(x)$

comment: Computes $y = Sx$

Send x from \widehat{L} to \widehat{R}

Compute $i^{(\widehat{L})} = K_{II}^{(\widehat{L})}x$

Compute $i^{(\widehat{R})} = K_{II}^{(\widehat{R})}x$

Compute $v_{\widehat{L}} = K_{\widehat{L}I}x$

Compute $v_{\widehat{R}} = K_{\widehat{R}I}x$

Solve $K_{\widehat{L}\widehat{L}}w_{\widehat{L}} = v_{\widehat{L}}$

Solve $K_{\widehat{R}\widehat{R}}w_{\widehat{R}} = v_{\widehat{R}}$

Compute $z^{(\widehat{L})} = K_{I\widehat{L}}w_{\widehat{L}}$

Compute $z^{(\widehat{R})} = K_{I\widehat{R}}w_{\widehat{R}}$

Compute $y^{(\widehat{L})} = i^{(\widehat{L})} - z^{(\widehat{L})}$

Compute $y^{(\widehat{R})} = i^{(\widehat{R})} - z^{(\widehat{R})}$

Send $y^{(\widehat{R})}$ from \widehat{R} to \widehat{L}

Compute $y = y^{(\widehat{L})} + y^{(\widehat{R})}$

return (y)

¹We used the C++ programming language in our implementation.

²This choice is somewhat arbitrary.

The new components in the method are (1) the S -MatVec and (2) the Recursive Schur Decomposition (RSD) preconditioner. The pseudocodes for these two components are given in Algorithms 3.1 and 3.2, respectively. In each of these two components, there are only two point-to-point communications between adjacent sub-domains; we also overlap these communications with computations. We used the standard Message Passing Interface (MPI) to handle these communications.

Algorithm 3.2: $\text{RSD}(K, f, h)$

comment: Solves $Ku = f$ approximately
comment: h is the height of the tree
if $h = 0$
 then Solve $Ku = f$
 Recursion: $v_L \leftarrow \text{RSD}(K_{LL}, f_L, (h - 1))$
 Recursion: $v_R \leftarrow \text{RSD}(K_{RR}, f_R, (h - 1))$
 Compute $g^{(\hat{L})} = K_{I\hat{L}}v_{\hat{L}}$
 Compute $g^{(\hat{R})} = K_{I\hat{R}}v_{\hat{R}}$
 Send $g^{(\hat{R})}$ from \hat{R} to \hat{L}
 Compute $g = f_I - g^{(\hat{L})} - g^{(\hat{R})}$
 else {
 Solve $Su_I = g$ approximately
 Send \hat{u}_I from \hat{L} to \hat{R}
 Compute $w_{\hat{L}} = K_{\hat{L}I}\hat{u}_I$
 Compute $w_{\hat{R}} = K_{\hat{R}I}\hat{u}_I$
 Solve $K_{\hat{L}\hat{L}}z_{\hat{L}} = w_{\hat{L}}$
 Solve $K_{\hat{R}\hat{R}}z_{\hat{R}} = w_{\hat{R}}$
 Compute $\hat{u}_{\hat{L}} = v_{\hat{L}} - z_{\hat{L}}$
 Compute $\hat{u}_{\hat{R}} = v_{\hat{R}} - z_{\hat{R}}$
 }
return (\hat{u})

3.3.1 Parallel Time Complexity

Let the number of true sub-domains be P , the number of unknowns per true sub-domain be M , and the number of S -MatVecs used to approximately solve Equation (3.2) be γ , the time (measured in terms of required floating point operations per processor) taken to apply the overall RSD algorithm once be $F(M, \gamma, P)$, and the time taken to apply the S -MatVec once be $G(M)$.

$G(M)$ is dominated by the time taken to solve (probably only approximately) the true sub-domain problem once. If this is done using a fast algorithm such as one Multigrid V-cycle then $G(M) = \mathcal{O}(M)$ and if this is done using a direct solver then $G(M) = \mathcal{O}(M^3)$.

Then, we have the recurrence relation: $F(M, \gamma, P) = F(M, \gamma, P/2) + \mathcal{O}(M) + \gamma G(M)$. By expanding this relation, we can show that $F(M, \gamma, P) = \mathcal{O}(\gamma G(M) \log P)$. Thus, the time taken to apply the overall RSD algorithm once grows logarithmically as the number of true sub-domains.

4 Results and Discussion

We tested our algorithm on four problems. The first is the Poisson equation shown in Equation (4.1).

$$-\nabla \cdot \nabla u = f \quad (4.1)$$

The second, Equation (4.2), and the third, Equation (4.3), are two system PDEs where the anisotropy in each variable differs; these equations were previously considered in [19]. The coupling between the u and v variables is weak for Equation (4.2) and very strong for Equation (4.3).

$$\begin{aligned} -\frac{u_{xx}}{100} - u_{yy} + \frac{v}{100} &= f \\ -\frac{u}{100} - v_{xx} - \frac{v_{yy}}{100} &= g \end{aligned} \quad (4.2)$$

$$\begin{aligned} -\frac{u_{xx}}{100} - u_{yy} + 100v &= f \\ -100u - v_{xx} - \frac{v_{yy}}{100} &= g \end{aligned} \quad (4.3)$$

The last, Equation (4.4), is the Navier-Lamé equation for elastostatics with the first (λ) and second (shear modulus, μ) Lamé parameters being 10 and 1, respectively.

$$-\nabla \cdot \nabla \vec{u} + 11\nabla \nabla \cdot \vec{u} = \vec{f} \quad (4.4)$$

All the equations were solved on two dimensional rectangular domains. In all cases, homogeneous Dirichlet boundary conditions were applied on all sides of the domain. In this paper, we used a regular grid with N nodes

N	$P = 8$			$P = 128$			$P = 2048$			$P = 32768$		
	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$
17	18	9	5	20	10	5	20	10	5	20	10	5
33	26	13	6	27	14	7	28	14	7	27	14	7
65	36	17	9	38	19	9	38	19	9	38	19	9
129	48	24	12	51	25	13	50	25	13	51	25	13

Table 1: Number of outermost Krylov iterations required to solve Equation (4.1).

N	$P = 8$			$P = 128$			$P = 2048$			$P = 32768$		
	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$
17	0.003	0.003	0.003	0.004	0.004	0.004	0.063	0.06	0.064	0.8	0.94	0.881
33	0.005	0.004	0.004	0.006	0.006	0.006	0.035	0.039	0.038	0.934	0.808	0.808
65	0.011	0.011	0.011	0.013	0.012	0.012	0.045	0.044	0.044	0.828	0.822	0.837
129	0.039	0.038	0.038	0.04	0.04	0.04	0.076	0.088	0.076	0.836	1.14	0.862

Table 2: Time (in seconds) taken for the setup phase for Equation (4.1).

in each dimension to mesh each true sub-domain. The equations were discretized using bi-linear finite elements. The resulting matrices were symmetric positive definite and diagonally dominant in some cases and unsymmetric and not diagonally dominant in others.

We performed several numerical experiments by solving each problem with various sets of parameters: N , P and γ . For each problem, we picked a random solution and used the method of manufactured solutions to construct the right hand side. Starting with a zero initial guess, we solved each problem to a relative tolerance of $1.0\text{e-}12$ in the 2-norm of the residual. We report the number of outermost Krylov iterations, β , (Tables 1, 4, 7 and 10) required to solve each problem as well as the time taken for the setup (Tables 2, 5, 8 and 11) and solve phases (Tables 3, 6, 9 and 12) in each case. The setup phase mainly involves creating the regular grid mesh, constructing the binary tree, creating the required MPI communicators, creating the matrices on each sub-domain and allocating memory for the vectors.

All the experiments were performed on the *Jaguar* supercomputer at Oak Ridge National Laboratory (ORNL). The architectural details for this supercomputer can be found in [15].

We can observe that the convergence rate of the algorithm does not deteriorate as the number of sub-domains increases. Theoretically, we estimated the time taken to apply the multi-level algorithm once to grow logarithmi-

N	$P = 8$			$P = 128$			$P = 2048$			$P = 32768$		
	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$
17	0.012	0.009	0.008	0.02	0.01	0.009	0.018	0.012	0.01	0.033	0.07	0.162
33	0.052	0.038	0.029	0.057	0.042	0.034	0.065	0.046	0.037	0.277	0.057	0.044
65	0.351	0.241	0.204	0.388	0.271	0.207	0.391	0.284	0.214	0.423	0.326	0.236
129	3.21	2.31	1.87	3.55	2.5	2.12	3.47	2.53	2.12	3.58	2.56	2.14

Table 3: Time (in seconds) taken for the solve phase for Equation (4.1).

N	$P = 8$			$P = 128$			$P = 2048$			$P = 32768$		
	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$
17	57	25	16	111	48	37	110	49	37	109	49	36
33	87	36	19	166	69	40	159	69	39	156	69	38
65	126	52	25	249	101	46	247	100	46	245	98	46
129	184	80	33	409	150	63	407	141	62	409	137	60

Table 4: Number of outermost Krylov iterations required to solve Equation (4.2).

N	$P = 8$			$P = 128$			$P = 2048$			$P = 32768$		
	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$
17	0.005	0.005	0.005	0.006	0.006	0.006	0.048	0.038	0.047	0.796	0.908	0.96
33	0.012	0.012	0.012	0.013	0.014	0.013	0.045	0.045	0.046	0.832	0.854	0.805
65	0.04	0.04	0.041	0.043	0.042	0.042	0.149	0.096	0.098	0.867	0.836	0.862
129	0.152	0.153	0.154	0.157	0.158	0.157	0.192	0.196	0.194	1.09	1.05	1.01

Table 5: Time (in seconds) taken for the setup phase for Equation (4.2).

N	$P = 8$			$P = 128$			$P = 2048$			$P = 32768$		
	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$
17	0.051	0.033	0.032	0.105	0.065	0.078	0.124	0.08	0.086	0.203	0.356	0.136
33	0.332	0.191	0.159	0.627	0.362	0.323	0.641	0.393	0.333	0.765	0.782	0.384
65	4.05	2.42	1.88	8.61	4.97	3.63	8.88	4.97	3.66	8.78	5.31	3.7
129	29.8	18.7	12.6	70.6	37.4	25.3	70.6	35.4	25	71.4	34.6	24.4

Table 6: Time (in seconds) taken for the solve phase for Equation (4.2).

N	P = 8			P = 128			P = 2048			P = 32768		
	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$
17	32	17	9	34	18	9	35	18	9	35	18	9
33	49	25	12	51	26	13	52	26	13	52	26	13
65	69	35	17	74	36	17	74	36	18	74	36	18
129	97	48	23	102	49	24	104	49	24	105	49	24

Table 7: Number of outermost Krylov iterations required to solve Equation (4.3).

N	P = 8			P = 128			P = 2048			P = 32768		
	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$
17	0.005	0.005	0.005	0.006	0.006	0.006	0.042	0.04	0.041	0.832	0.848	0.831
33	0.012	0.012	0.012	0.013	0.013	0.014	0.047	0.047	0.047	0.801	0.861	0.821
65	0.04	0.04	0.04	0.042	0.042	0.042	0.097	0.096	0.098	0.815	0.843	0.882
129	0.153	0.153	0.153	0.157	0.158	0.157	0.194	0.194	0.193	1.04	1.05	1.04

Table 8: Time (in seconds) taken for the setup phase for Equation (4.3).

N	P = 8			P = 128			P = 2048			P = 32768		
	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$
17	0.03	0.023	0.02	0.045	0.027	0.021	0.041	0.031	0.024	0.096	0.05	0.071
33	0.187	0.139	0.107	0.204	0.148	0.121	0.228	0.16	0.126	0.707	0.175	0.147
65	2.28	1.67	1.31	2.64	1.86	1.42	2.66	1.88	1.51	2.71	1.91	1.53
129	16.2	11.6	9.07	18.4	12.9	10.3	18.8	12.9	10.4	19	13	10.4

Table 9: Time (in seconds) taken for the solve phase for Equation (4.3).

N	P = 8			P = 128			P = 2048			P = 32768		
	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$
17	39	18	10	45	21	13	45	21	13	45	21	13
33	57	26	13	63	29	15	63	29	15	63	29	15
65	79	36	17	86	41	19	86	41	19	85	41	19
129	112	52	24	119	56	26	118	56	26	118	56	26

Table 10: Number of outermost Krylov iterations required to solve Equation (4.4).

N	$P = 8$			$P = 128$			$P = 2048$			$P = 32768$		
	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$
17	0.005	0.005	0.005	0.006	0.006	0.006	0.041	0.067	0.041	0.778	0.77	0.93
33	0.012	0.012	0.012	0.013	0.013	0.013	0.046	0.046	0.046	0.919	0.817	0.804
65	0.04	0.04	0.041	0.042	0.043	0.042	0.096	0.096	0.098	0.823	0.836	0.868
129	0.152	0.152	0.155	0.157	0.158	0.158	0.195	0.208	0.195	1.06	1.06	1.05

Table 11: Time (in seconds) taken for the setup phase for Equation (4.4).

N	$P = 8$			$P = 128$			$P = 2048$			$P = 32768$		
	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$	$\gamma = 2$	$\gamma = 4$	$\gamma = 8$
17	0.036	0.024	0.022	0.05	0.031	0.029	0.055	0.034	0.033	0.278	0.121	0.082
33	0.217	0.144	0.117	0.247	0.166	0.137	0.297	0.175	0.143	0.484	0.193	0.152
65	2.61	1.71	1.32	3.04	2.1	1.57	3.07	2.11	1.59	3.12	2.14	1.6
129	18.4	12.4	9.43	21.3	14.6	11.1	21.2	14.7	11.2	21.4	14.7	11.1

Table 12: Time (in seconds) taken for the solve phase for Equation (4.4).

cally with P but, in practice the time taken for this seems to be roughly independent of P . This suggests that the constants in the complexity estimate are small.

In this work, we did not use any preconditioner for solving (approximately) the Schur complement systems. As a result, the convergence rate for the inner Krylov subspace method used to solve (approximately) these systems deteriorates as the number of unknowns per sub-domain increases. Consequently, if the number of inner Krylov iterations is kept fixed then the quality of the overall multi-level preconditioner deteriorates as M increases. Hence, for a fixed γ the number of outermost Krylov iterations required to solve the problem increases as M increases.

For a fixed M , β appears to be inversely proportional to γ . Since the time taken to apply the overall preconditioner once is directly proportional to γ , one might expect the time taken for the solve phase to be unaffected by changing γ . Nevertheless, increasing γ tends to reduce the time taken for the solve phase. This can be attributed to the fact that by reducing β , we are reducing the number of global synchronous communications (e.g., MPI reductions to compute inner-products of vectors) involving all the processors.

5 Conclusions

We presented a parallel algorithm using multi-level domain decomposition for solving sparse linear systems of equations. We tested the performance of this algorithm on several PDEs. In all cases, the performance of the algorithm did not deteriorate as the number of sub-domains increased. To our knowledge, this is the first domain decomposition algorithm that has scaled to such a large number of sub-domains. Also, the running time (including the setup phase) for the solver is small. This is a very desirable characteristic particularly in the context of solving nonlinear and transient problems as these involve solving several linear systems of equations.

We can improve the performance of the overall algorithm by using a good preconditioner for the inner Krylov subspace method. Another useful improvement would be to use more processors per sub-domain. By allowing a variable number of processors per sub-domain, we could make the overall algorithm well load balanced even when the distribution of the work load across the domains is not uniform. Finally, we could extend the algorithm to solve three dimensional problems on more general domains by working with two dimensional and three dimensional domain decompositions using quadtrees and octrees instead of binary trees.

Acknowledgements

This work was carried out as part of the Advanced Multi-Physics (AMP) [4] code development. The development of the AMP code was funded by the Nuclear Energy Advanced Modeling and Simulation (NEAMS) program of the U.S. Department of Energy Office of Nuclear Energy, Advanced Modeling and Simulation Office. The AMP code is open-source and available by contacting the author, or available with a single-user license through the Radiation Safety Information Computational Center (RSICC) at Oak Ridge National Laboratory as Package C793.

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

- [1] Satish Balay, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. “PETSc users manual”. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2004.
- [2] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. “PETSc home page”, 2001. <http://www.mcs.anl.gov/petsc>.
- [3] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. “Efficient management of parallelism in object oriented numerical software libraries”. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163 – 202. Birkhäuser Press, 1997.
- [4] Kevin T. Clarno, Bobby Philip, William K. Cochran, Rahul S. Sampath, Srikanth Allu, Pallab Barai, Srdjan Simunovic, Larry J. Ott, Sreekanth Pannala, Phani Nukala, Gary A. Dils, Bogdan Mihaila, Cetin Unal, Gokhan Yesilyurt, Jung Ho Lee, James E. Banfield, and Guillermo Ivan Maldonado. “The AMP (Advanced Multi-Physics) nuclear fuel performance code”. Technical Report ORNL/TM-2011/42, Oak Ridge National Laboratory, 2011.
- [5] T. Corman, C. Leiserson, and R. Rivest. “*Introduction to algorithms*”. MIT Press, 1990.
- [6] C. R. Dohrmann. “An approximate BDDC preconditioner”. *Numerical Linear Algebra with Applications*, 14(2):149–168, 2007.
- [7] Charbel Farhat, Michel Lesoinne, Patrick LeTallec, Kendall Pierson, and Daniel Rixen. “FETI-DP: A dual-primal unified FETI method part I: A faster alternative to the two-level FETI method”. *International Journal For Numerical Methods in Engineering*, 50(7):1523–1544, 2001.
- [8] J. Gaidamour and P. Henon. “A parallel direct/iterative solver based on a Schur complement approach”. In *IEEE International Conference on Computational Science and Engineering*, 2008.
- [9] L. Grinberg and G.M. George Em Karniadakis. “A scalable domain decomposition method for ultra-parallel arterial flow simulations”. *Communications in Computational Physics*, 4(5):1151–1169, 2008.

- [10] Pascal Henon and Yousef Saad. “A parallel multistage ILU factorization based on a hierarchical graph decomposition”. *SIAM Journal on Scientific Computing*, 28(6), 2006.
- [11] D.D. Joseph, R. Bai, K.P. Chen, and Y.Y. Renardy. “Core-annular flows”. *Annual Review of Fluid Mechanics*, 29:65–90, 1997.
- [12] Jan Mandel. “Balancing domain decomposition”. *Communications in Numerical Methods in Engineering*, 9(3):233–241, 1993.
- [13] Murat Manguoglu, Ahmed H. Sameh, Tayfun E. Tezduyar, and Sunil Sathe. “A nested iterative scheme for computation of incompressible flows in long domains”. *Computational Mechanics*, 43, 2008.
- [14] Tarek P.A. Mathew. “*Domain decomposition methods for the numerical solution of partial differential equations*”. Springer, 2008.
- [15] NCCS. “Jaguar’s system architecture”. <http://www.nccs.gov/computing-resources/jaguar>.
- [16] D. Olander. “Nuclear fuels - present and future”. *Journal of Nuclear Materials*, 389(1), 2009.
- [17] D.R. Olander. “*Fundamental aspects of nuclear reactor fuel elements*”. Technical Information Center, Office of Public Affairs, Energy Research and Development Administration, 1976.
- [18] M.P. Paidoussis. “*Fluid-structure interactions: slender structures and axial flow*”, volume 1. Academic Press, 1998.
- [19] Bobby Philip and Timothy P. Chartier. “Adaptive algebraic smoothers”. *Journal of Computational and Applied Mathematics*, 236, 2012.
- [20] Yousef Saad. “*Iterative methods for sparse linear systems*”. Society for Industrial and Applied Mathematics, 2 edition, 2003.
- [21] Barry Smith, Petter Bjorstad, and William Gropp. “*Domain decomposition: parallel multilevel methods for elliptic partial differential equations*”. Cambridge University Press, 2004.
- [22] Andrea Toselli and Olof Widlund. “*Domain decomposition methods - algorithms and theory*”. Springer, 2005.

- [23] X.P. Zhang, C. Rehtanz, and B. Pal. “*Flexible AC transmission systems: modelling and control*”. Springer, 2006.